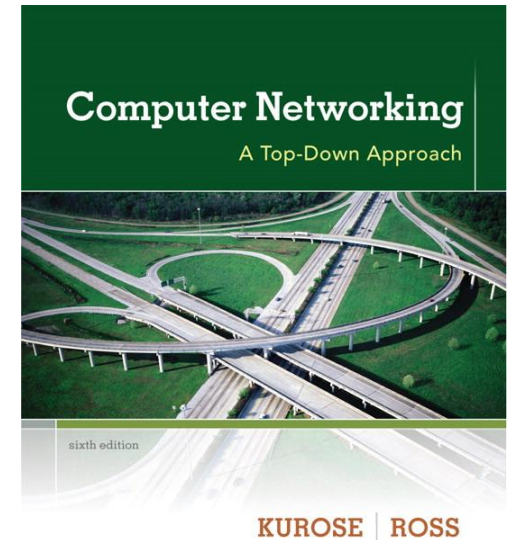


Chapter 3

Transport Layer



*Computer
Networking: A Top
Down Approach*
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 3: Transport Layer

our goals:

- ❖ understand principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- ❖ learn about Internet transport layer protocols:
 - UDP: connectionless transport
 - TCP: connection-oriented reliable transport



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

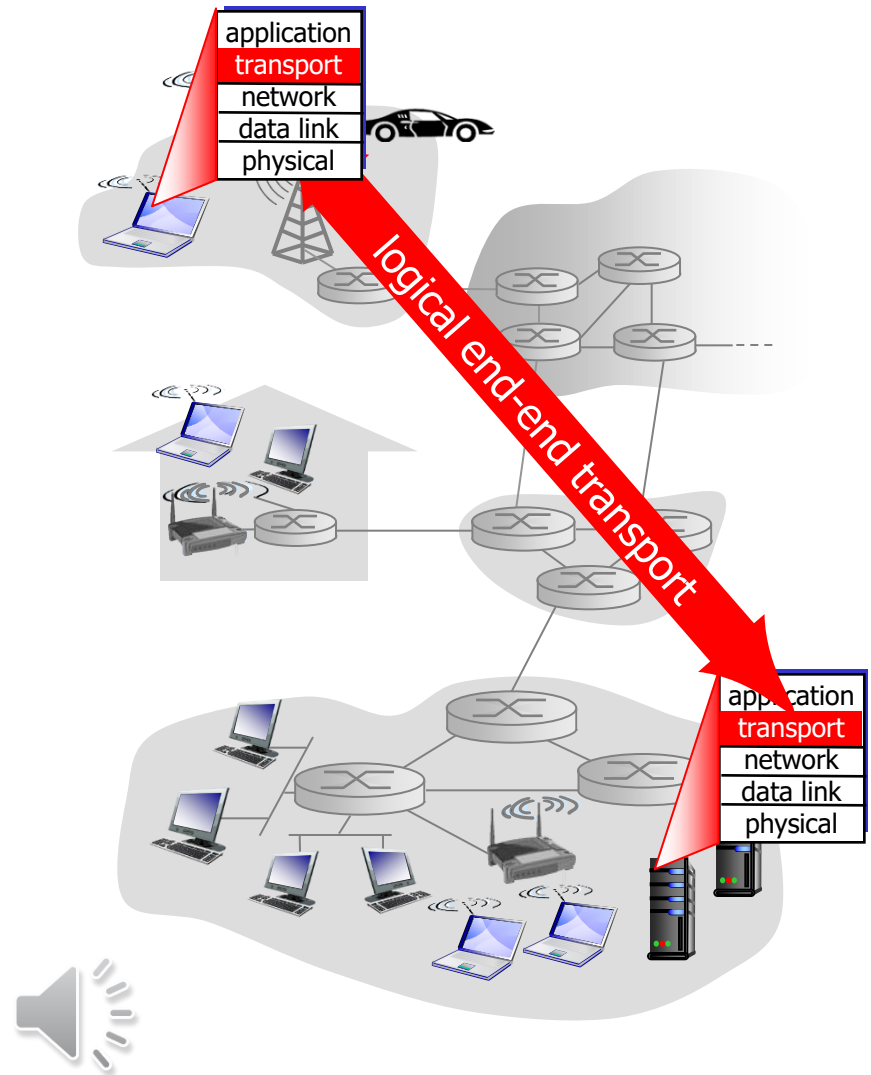
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management



Transport services and protocols

- ❖ provide *logical communication* between app processes running on different hosts
- ❖ transport protocols run in end systems
 - send side: breaks app messages into *segments*, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- ❖ more than one transport protocol available to apps
 - Internet: TCP and UDP



Transport vs. network layer

- ❖ *network layer*: logical communication between hosts
- ❖ *transport layer*: logical communication between processes
 - relies on, enhances, network layer services



Internet transport-layer protocols

- ❖ reliable, in-order delivery (TCP)

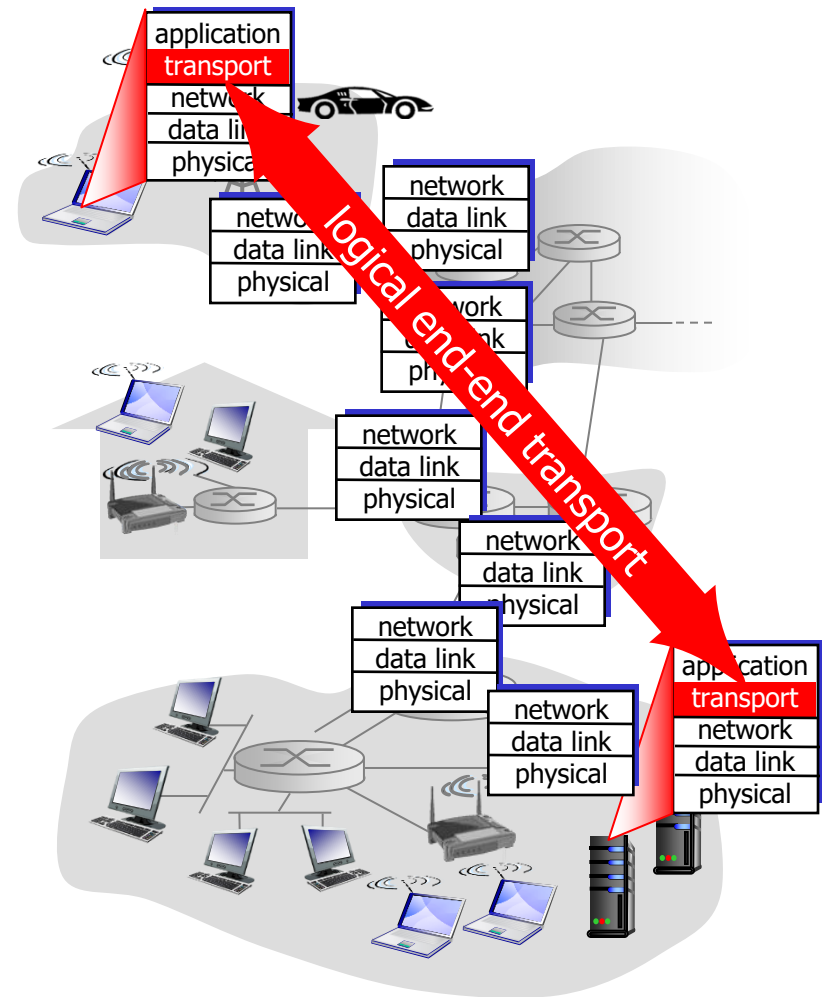
(TCP ensures that data is delivered from sending process to receiving process, correctly and in order.)

- congestion control
- flow control
- connection setup

- ❖ unreliable, unordered delivery: UDP

- ❖ services not available:

- delay guarantees
- bandwidth guarantees



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

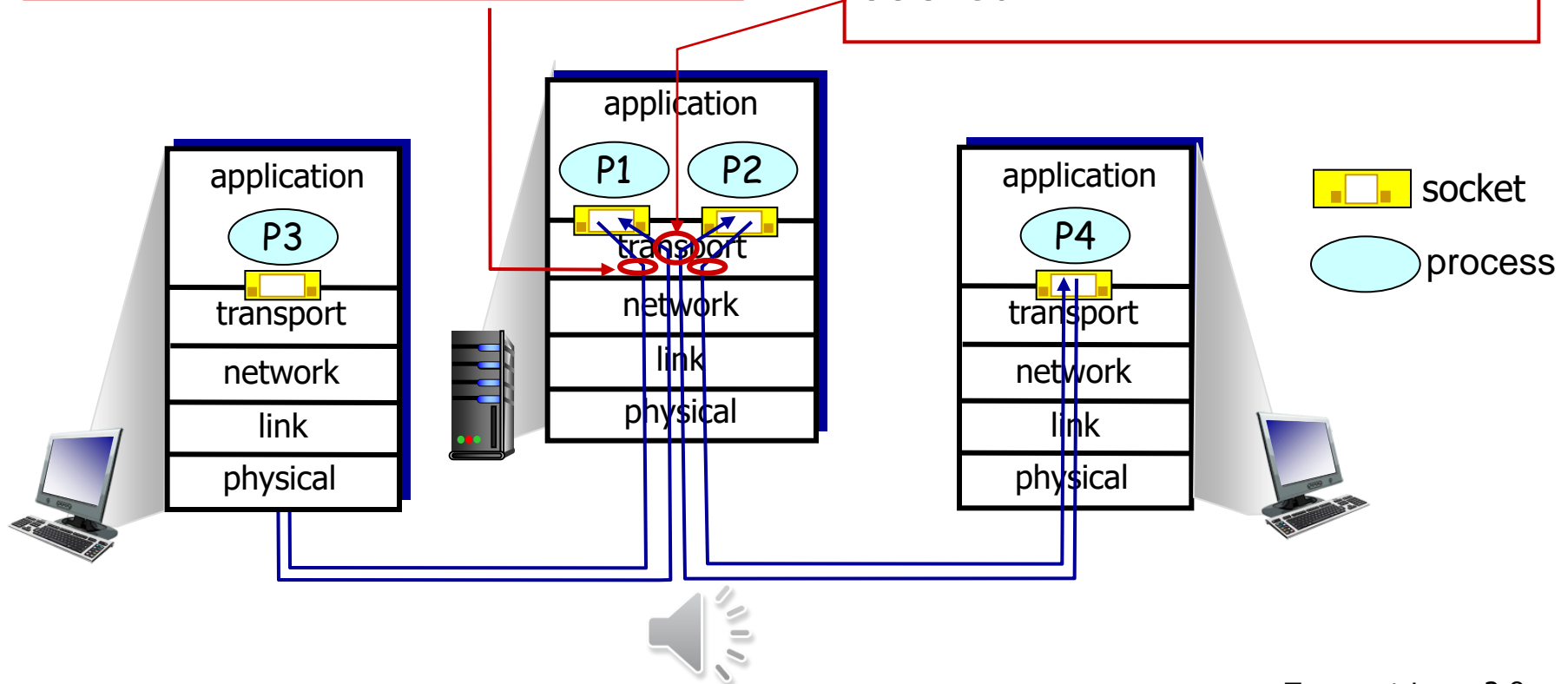
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

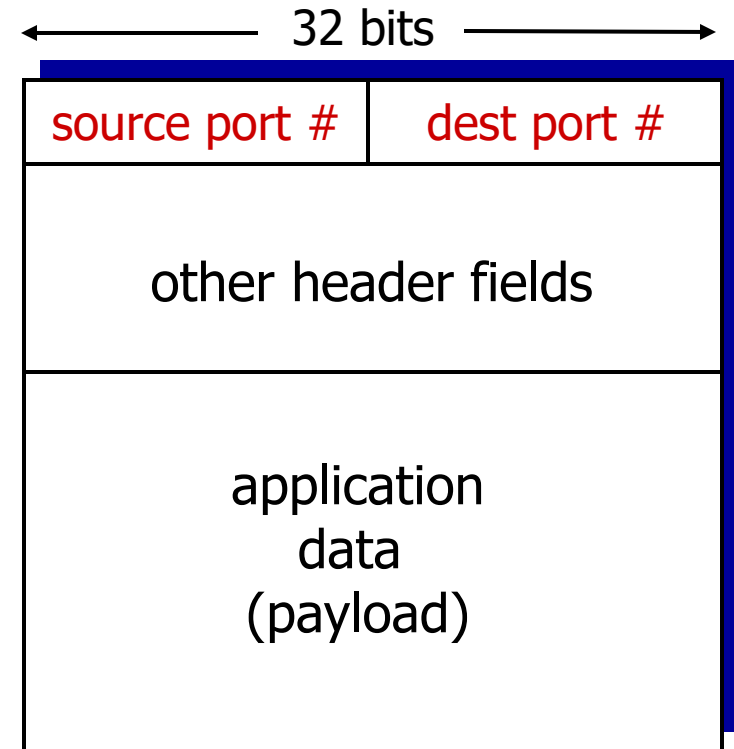
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

- ❖ host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries one transport-layer segment
 - each segment has source, destination port number
- ❖ host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

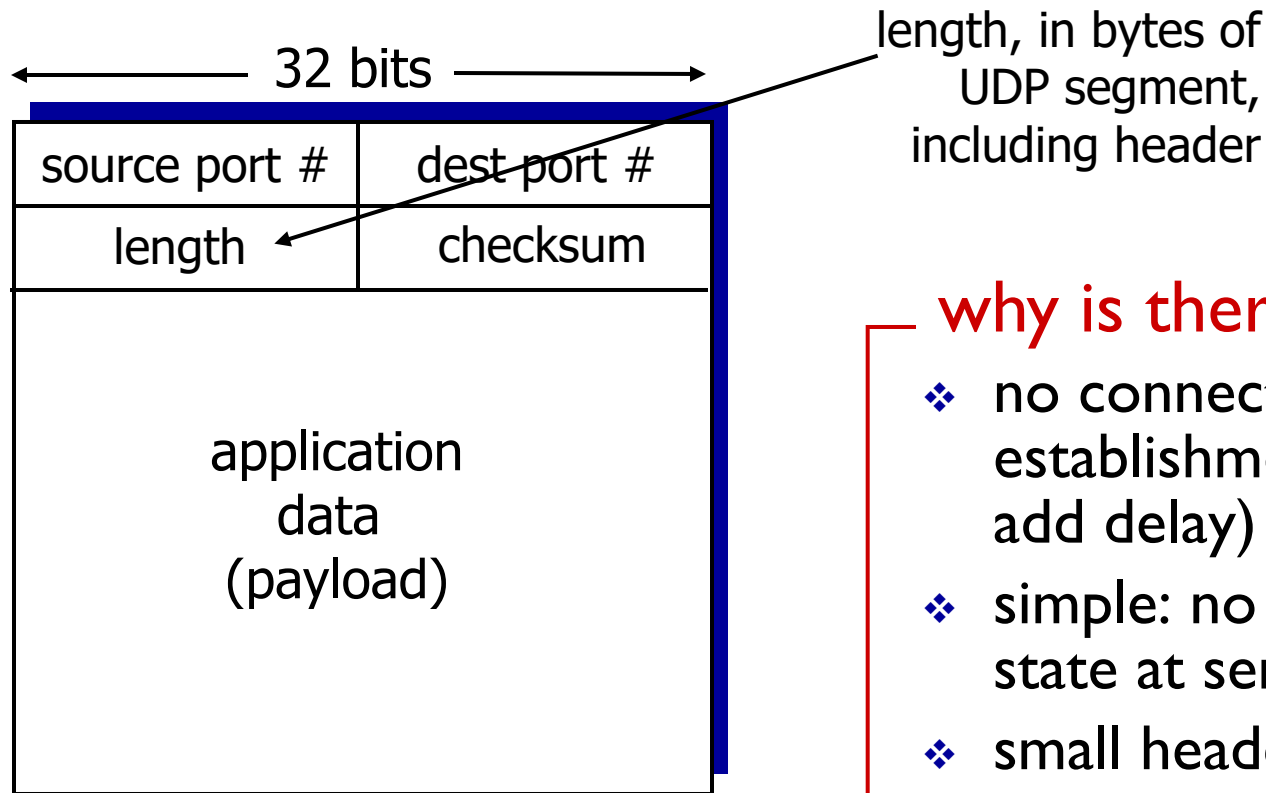
UDP: User Datagram Protocol [RFC 768]

- ❖ “best effort” service, UDP segments may be:
 - lost
 - delivered out-of-order to app
- ❖ *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others



- ❖ UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
- ❖ reliable transfer over UDP:
 - add reliability at application layer
 - application-specific error recovery!

UDP: segment header



UDP segment format

why is there a UDP?

- ❖ no connection establishment (which can add delay)
- ❖ simple: no connection state at sender, receiver
- ❖ small header size
- ❖ no congestion control



UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

sender:

- ❖ treat segment contents, including header fields, as sequence of 16-bit integers
- ❖ checksum: addition (one's complement sum) of segment contents
- ❖ The one's complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s
- ❖ sender puts checksum value into UDP checksum field



receiver:

- ❖ compute checksum of received segment
- ❖ check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected. *But maybe errors nonetheless? More later*

Internet checksum: example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
<hr/>																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

❖ point-to-point:

- one sender, one receiver

❖ reliable, in-order *byte stream*:

- no “message boundaries”



❖ pipelined:

- TCP congestion and flow control set window size

❖ full duplex data:

- bi-directional data flow in same connection

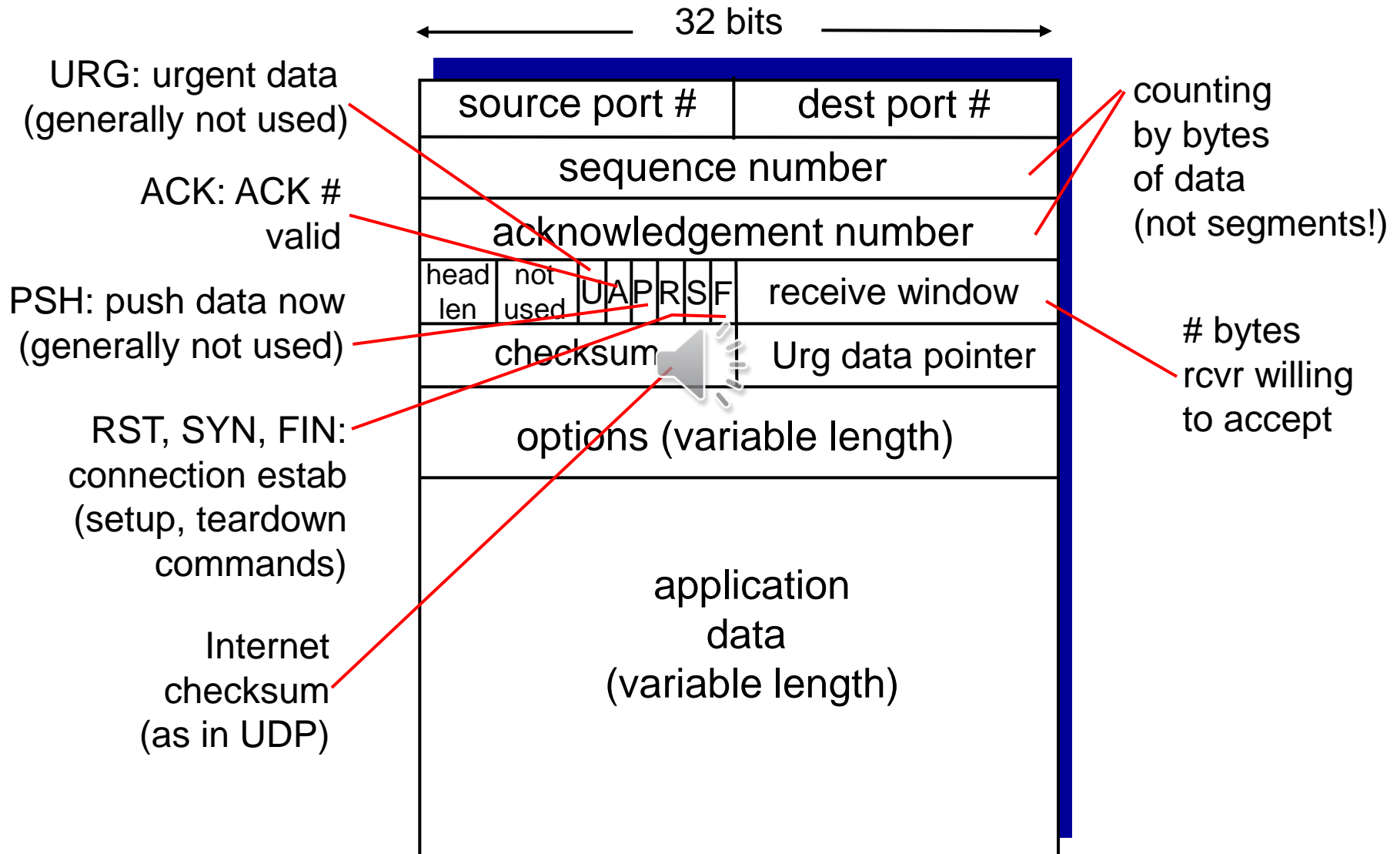
❖ connection-oriented:

- handshaking (exchange of control msgs) initializes sender, receiver state before data exchange

❖ flow controlled:

- sender will not overwhelm receiver

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

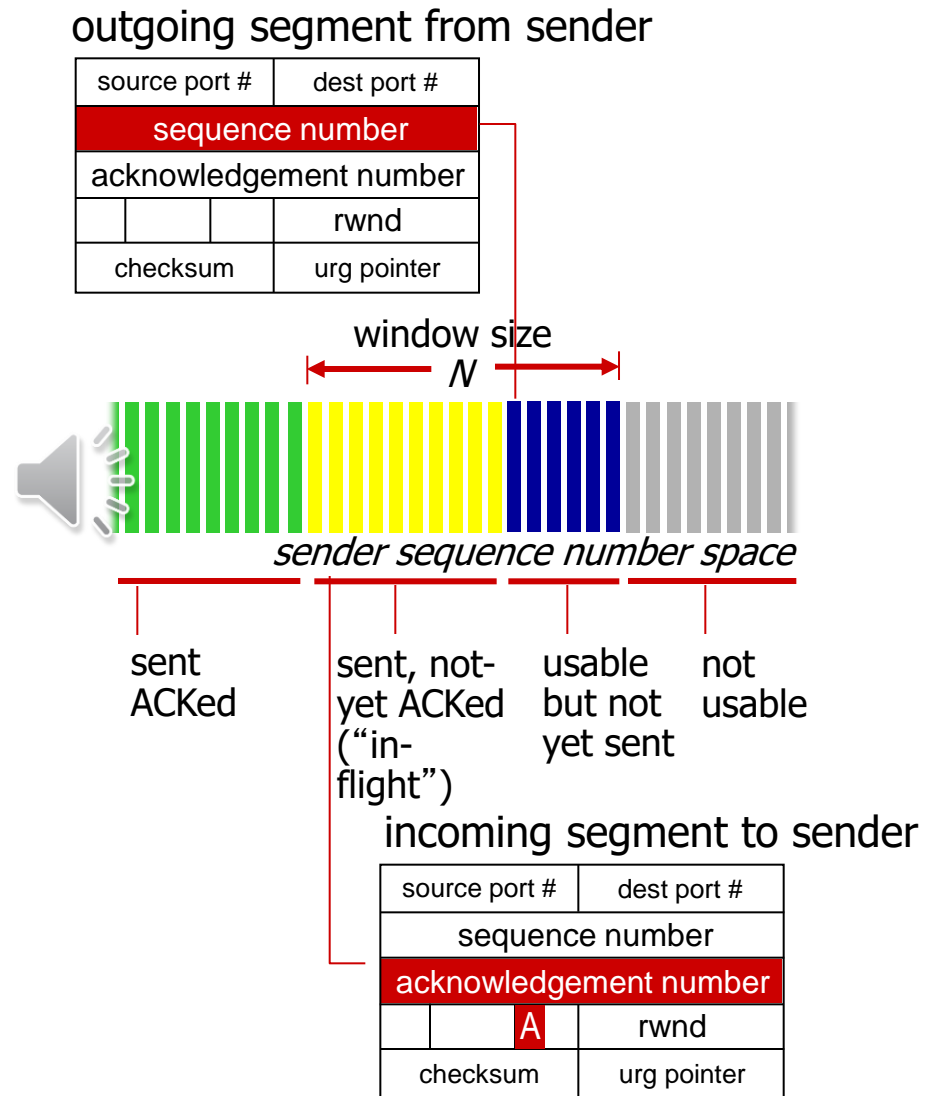
- byte stream “number” of first byte in segment’s data

acknowledgements:

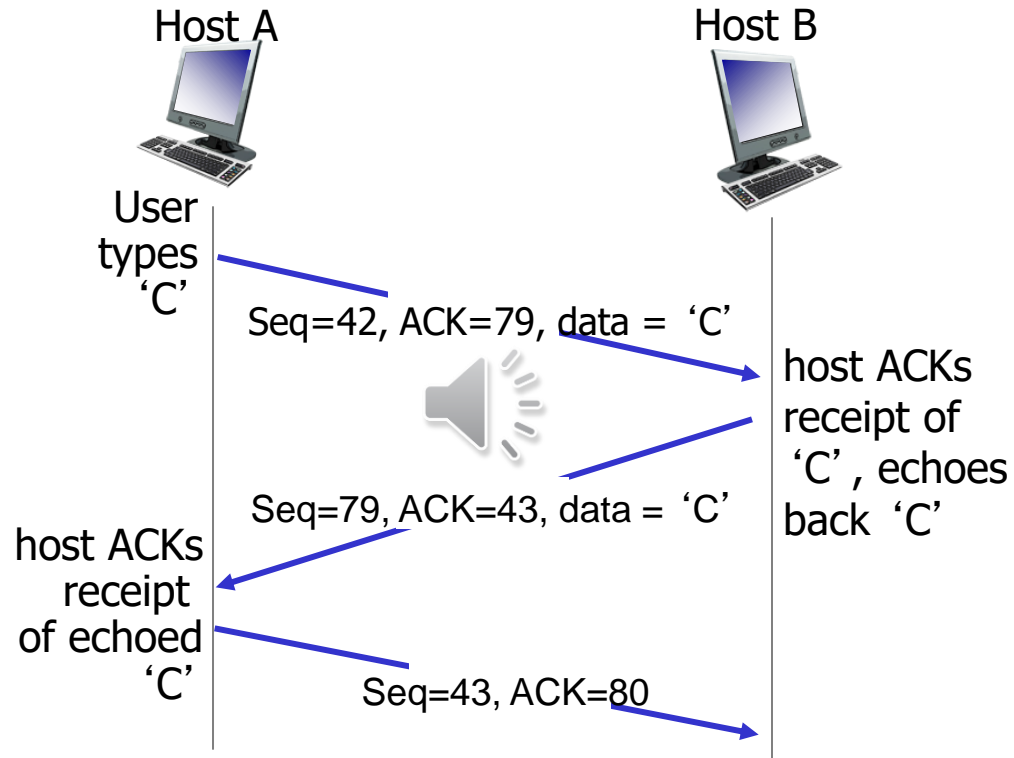
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A:** TCP spec doesn’t say,
- up to implementor



TCP seq. numbers, ACKs



simple telnet scenario

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

TCP reliable data transfer

- ❖ TCP creates rdt service on top of IP's unreliable service

- pipelined segments
- cumulative acks
- single retransmission timer

- ❖ retransmissions triggered by:

- timeout events
- duplicate acks

 let's initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

TCP sender events:

data rcvd from app:

- ❖ create segment with seq #
- ❖ seq # is byte-stream number of first data byte in segment
- ❖ start timer if not already running

timeout:

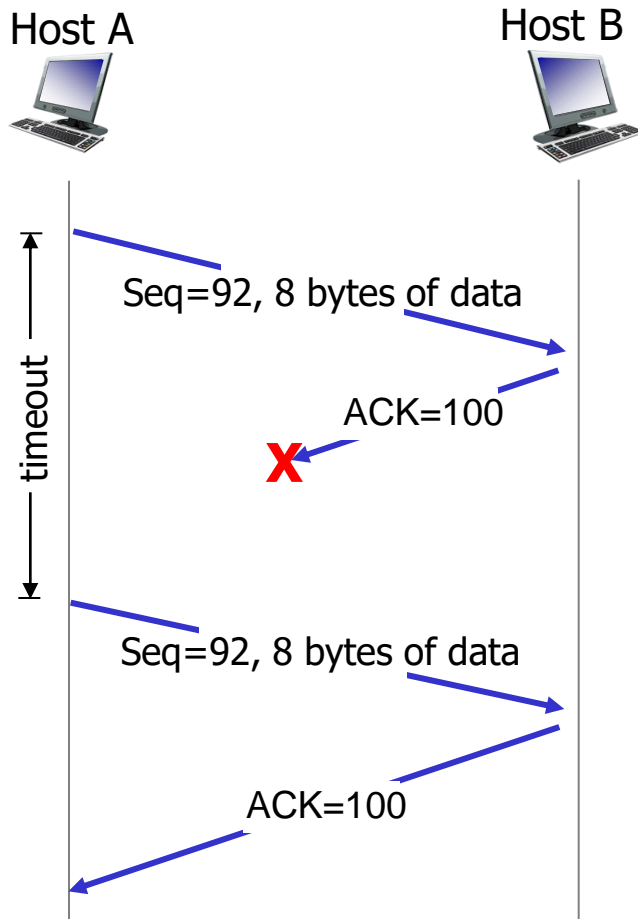
- ❖ retransmit segment that caused timeout
- ❖ restart timer

ack rcvd:

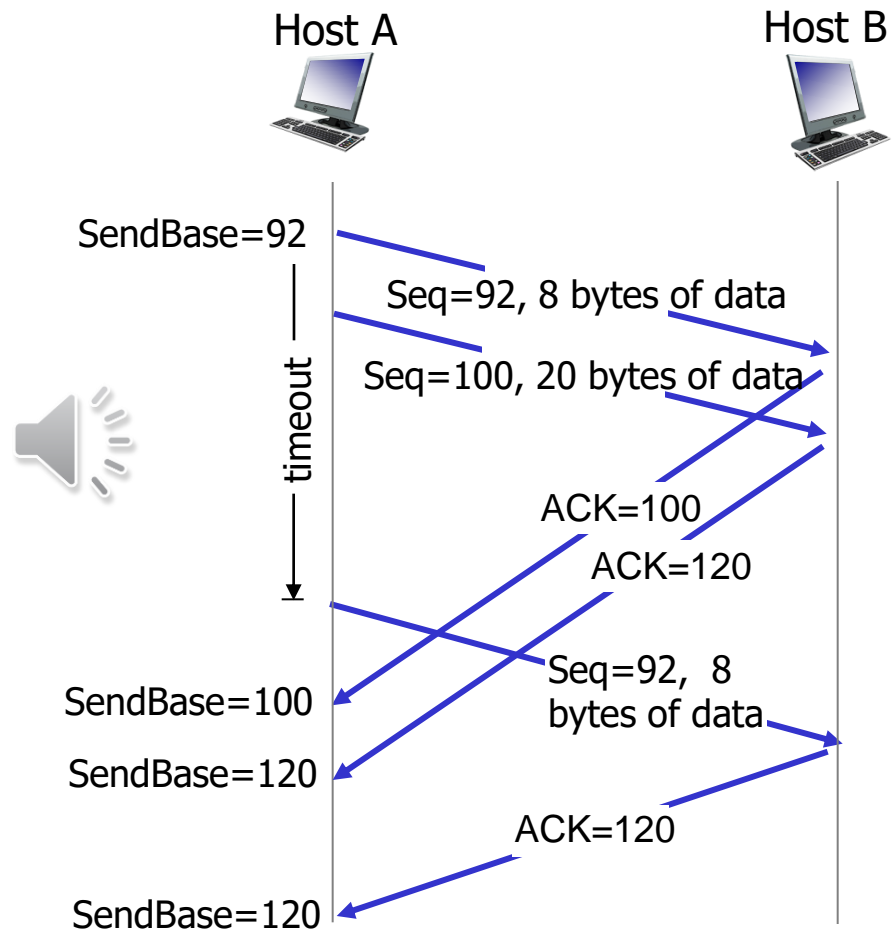


- ❖ if ack acknowledges previously unacked segments
 - update what is known to be ACKed
 - start timer if there are still unacked segments

TCP: retransmission scenarios



lost ACK scenario



premature timeout

TCP: retransmission scenarios

